SIR Programming in R Notebook

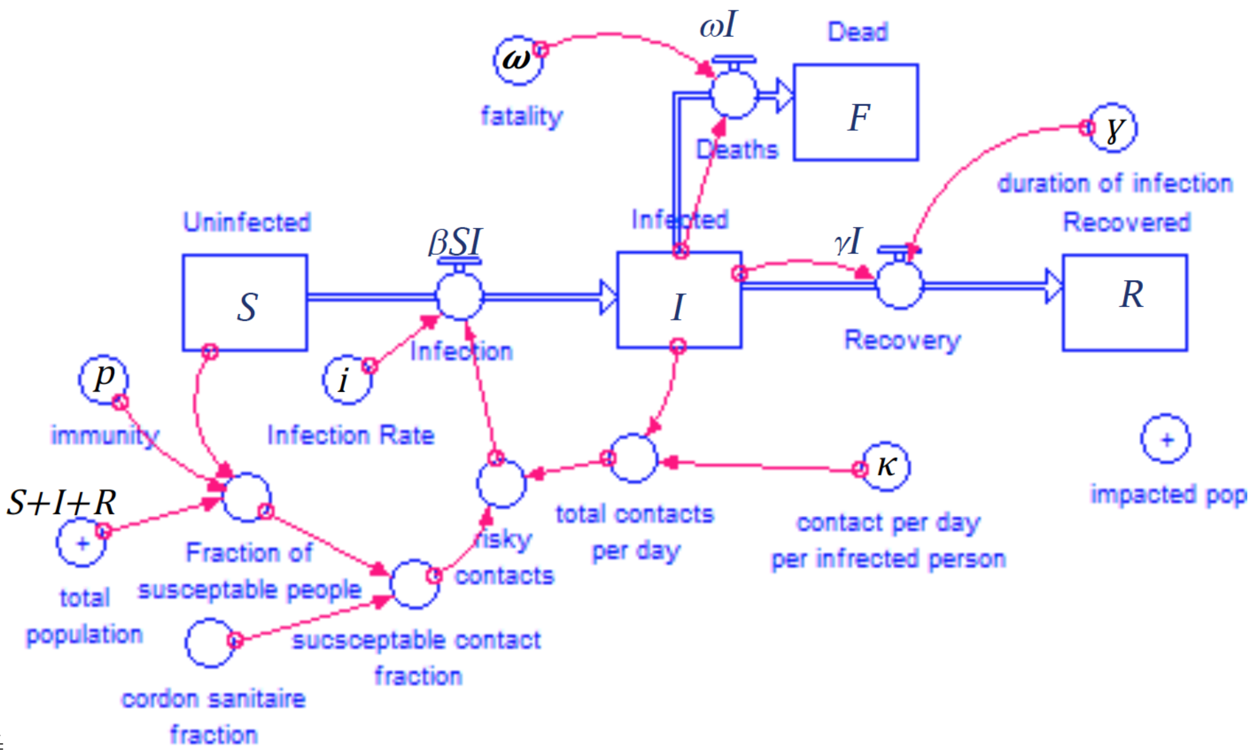
This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Here we are representing an SIR Model (Susceptable vs Infected vs Recovered [vs Fatalities])

This is to demonstarte the programming elements of R rather than the statistical elements of R since R is frequently now being used as an open-source (read "free") resoruce for programming.

Here is a sketch of our SIRF model



#

This creates four equations whcih are based on the "Kermack-McKendrick Equations," an idealization of a disease system model.

$$\frac{dS}{dt} = -\beta S I$$

$$\frac{dI}{dt} = +\beta S I - \gamma I - \omega I$$

$$\frac{dR}{dt} = +\gamma I$$

$$\frac{dF}{dt} = +\omega I$$

To do this we will break this process down into several sub-sections

1) We will do time control (set up the period of our simulation) R lets us do some tricks here
2) We will create the variables for which we will preduct (our Sick, Healthy, Recovered and Dead people)
3) We then will set up the parameters that describe how our disease will behave
4) We will integrate forward in time with a basic Euler's "solver" for our system of equations.
5) Then plot our output

First let's set up the time control of our simulation.

```
##############################################################################
#
# time control section
#

t_0     <-  0.00  # start time in days
t_f     <- 30.00  # end time in days
delta_t <-  0.10  # time step in R

N       <- (t_f - t_0) / delta_t + 1 # number of timesteps

time    <- seq(from = t_0,       # creates an array from ...
               to   = t_f,       # ... to  ...
               by   = delta_t)   # ... using ain increment of ...

N_alt   <- length(time) # another way to get the total number of
                        # timesteps

# in R we also have an opportunity to create a genuine time string
#   with a date.

# here is how you do it.

# step 1:  Create a start date the date string is
#          YYYY-MM-DD HH:mm:SS.fraction of sec
#          You can even use a time zone

start_assessed_date <- as.POSIXct('2016-01-09 00:00:00.0', tz = "MST")
```

```
# step 2: If you have already created a generic time array in units of
#         "days" starting at zero you can just add time to your "start"
#          date.  If's it's in hours you can divide it by 24.  If it's
#          in months you're on your own.  That gets tricky.

calendar_date      <- time + start_assessed_date

#
################################################################################
```

With our time set up we can now create a set of "players" in this game

Susceptables (people who have not acquired the illness)

Infected (people who have acquired the illness and in this scenario are showing symptions AND are contagious)

Recovered (people who have recovered from the illness and are no longer infectious)

Fatalities (people who had the illness but are no longer breathing or doing much of anything else)

```
################################################################################
#
# variable creation
#

S <- time * 0.0  # empty Susceptable Array
I <- time * 0.0  # empty Infected Array
R <- time * 0.0  # empty Recovered Array
F <- time * 0.0  # empty Fatalities Array

# and initial values are nice.
# here we will have a healthy "campus" with one person
# coming back from Christmas Break with a "souvenier"

S[1] <- 9999.0  # Potential Future Patients
I[1] <-    1.0  # Patient Zero (the first to be infected in a population)
R[1] <-    0.0  # Potential Recovered Patients
F[1] <-    0.0  # Potential Casualties

#
################################################################################
```

Now we can create the rules for our outbreak

We will need to have the

1) the likely number of people who an infected person comes into contact per unit time (i)
2) the probability of infection when a truly vulerable person is exposed per unit time (kappa)
3) the probability of recovery per unit time for a given infected person (gamma)
4) the probability per unit time of an infected person dying from the disesase (omega)

Beta from the equation system is a little more complex than the original case and is a combination of several of these terms

$$\beta = \frac{i\kappa(1-p)}{S+I+R}$$

If we want to include factors like artificial or natural immunity (p) we can include that too

We also can create concepts like the Criticial Herd Immunity (pc) and also the the Basic Reproduction Number (Ro)

```
################################################################################
#
# setting disease paramters
#

i     <- 0.25   # probability of infection on exposure per unit time

kappa <- 6.00   # number of people with whom an infected person has contact
                # per unit time
gamma <- 0.50   # probability of an infected person recovering per unit time

omega <- 0.10   # probability of an infected person dying per unit time

p     <- 0.00   # percent of the population with immunity going into the
                # disease outbreak

#
################################################################################
```

With this we can now put all of our pieces of the model together.

But first let's create some support variables.

The first is the Basic Reproduction Number (Ro) or the typical expected number of people that are then infected by a single given case.

For Ro < 0, then the disease should be expected not spread through a population For Ro > 0, then the disease should be expected to spread through a population.

$$R_o = \frac{i\kappa}{\gamma + \omega}$$

The other is the critical herd immunity by which the growth of the infection is below zero.

$$p_o = 1 - \frac{1}{R_o}$$

```
################################################################################
#
# diagnostic disease parameters
#

Ro <- i * kappa / ( gamma + omega ) # Basic Reproductive Number

pc <- 1 - 1 / Ro                    # threshold herd immunity

print(paste(" Ro Factor: ",
            Ro
            )
      )
```

```
## [1] " Ro Factor:  2.5"
```

```
print(paste(" Threshold Herd Immunity: ",
            pc
            )
      )
```

```
## [1] " Threshold Herd Immunity:  0.6"
```

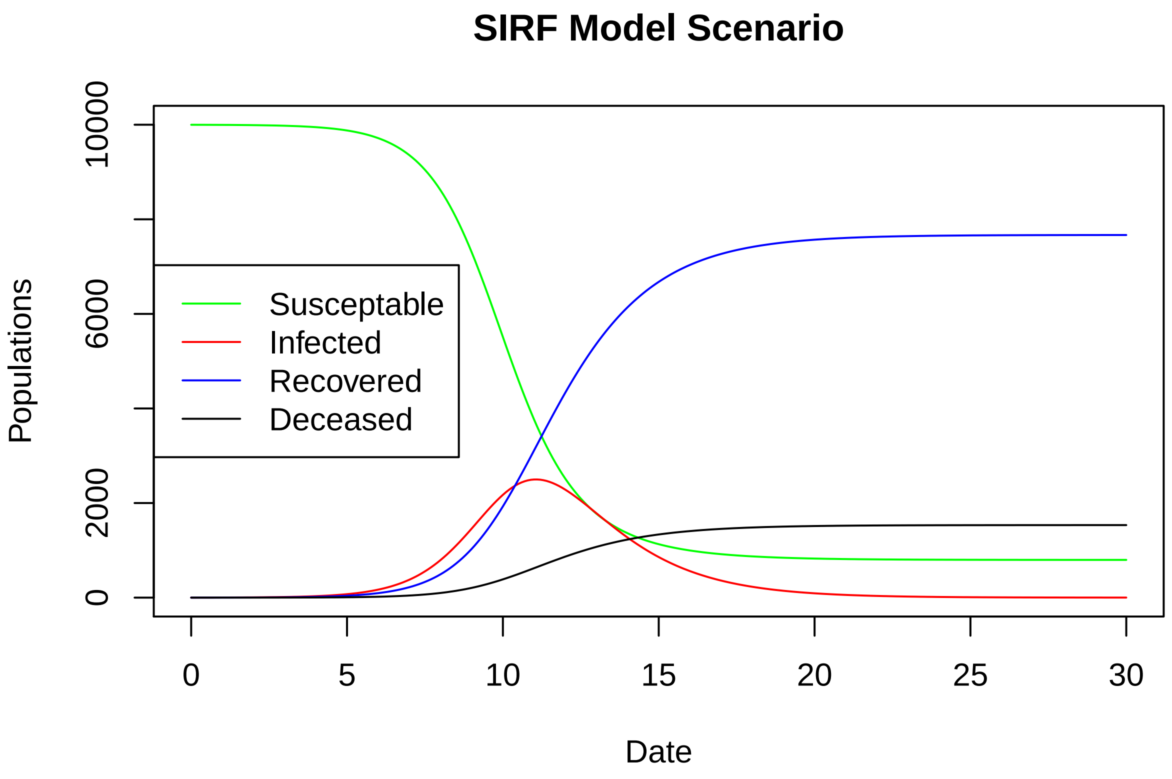
```
#
################################################################################
```

And with the parameters of the model established, we can now create the equations needed to move forward in time. Here we will be using a simple Euler's Method.

```
################################################################################
#
# marching forward in time using Euler's Method
#

for ( m in 2:N )
{

  #
  # set beta parameter for this time step
  #

  beta = (1-p) * kappa * i / ( S[m-1]+I[m-1]+R[m-1] )

  #
  # create our rates of change at this time step
  #

  dSdt <- - beta * S[m-1] * I[m-1]
  dIdt <-   beta * S[m-1] * I[m-1] - gamma * I[m-1] - omega * I[m-1]
  dRdt <-   gamma * I[m-1]
  dFdt <-   omega * I[m-1]

  #
  # move forward in time
  #

  S[m] <- S[m-1] + dSdt * delta_t
  I[m] <- I[m-1] + dIdt * delta_t
  R[m] <- R[m-1] + dRdt * delta_t
  F[m] <- F[m-1] + dFdt * delta_t

  #
  # A patch to acknowlege that while you can have
  # negative personalities, you can't have negative
  # people
  #

  if ( S[m] < 0 )
  {
    S[m] <- 0
  }
  #
  # but this is a prettier way to do it
  #
  #  We don't need it for fatalities but there could be a scenario
  #     where recovered populations become sucsceptable once again
  #     and some of you psychopaths may want to start a zombie
  #     apocalypse.  I don't judge.
  #

  I[m] = max( I[m], 0.0 ) # "flooring" populations to zero
  R[m] = max( R[m], 0.0 ) # "flooring" populations to zero
  F[m] = max( F[m], 0.0 ) # "flooring" populations to zero

}

#
############################################################################
```

Now is a good time to plot our results.

```
############################################################################
#
# graphical output
#

# draw the primary plot

plot(x    = time         ,             # x-values
     y    = S,                         # y-values
     main = "SIRF Model Scenario",     # main title string
     xlab = "Date",                    # x-labels
     ylab = "Populations",             # y-labels
     col  = "green",                   # plot color
     type = "l",                       # use line plots only
     lty  =  1,                        # solid line
     ylim = c(0, max((S+I+R+F)))       # y-axis range
     )

# draw the three extra lines

lines(x    = time,     # x value
      y    = I,        # y value
      col  = "red",    # line color
      lty  = 1         # make a solid line
      )

lines(x    = time,     # x value
      y    = R,        # y value
      col  = "blue",   # line color
      lty  = 1         # make a solid line
      )

lines(x    = time,     # x value
      y    = F,        # y value
      col  = "black",  # line color
      lty  = 1         # make a solid line
      )



legend(x      = "left",            # legend location
       legend = c("Susceptable",    # legend text
                  "Infected",
                  "Recovered",
                  "Deceased"),
       col    = c("green",          # line/marker color
                  "red",
                  "blue",
                  "black"),      # make solid lines
       lty    = c(1, 1, 1)
       )
```



```
#
################################################################################
```

Here are some things you can try to explore.

1) Try to emulate various disease Ro's from the projects introductory powerpoint slide.
2) Explore how herd immunity will impact the total number of cases.
3) Create a second value of Kappa that will reduce the number of Infected contacts once the total number of people with the disease reaches a given value (that will require an if-then-block)